universität freiburg

Algorithm Theory – WS 2024/25

Chapter 7 : Randomization I

Introduction / Primality Test / Randomized Quicksort

Fabian Kuhn
Dept. of Computer Science
Algorithms and Complexity

Randomization

Randomized Algorithm:

- An algorithm that uses (or can use) random coin flips in order to make decisions

We will see: randomization can be a powerful tool to

- Make algorithms faster
- Make algorithms simpler
- Make the analysis simpler
  - Sometimes it's also the opposite…
- Allow to solve problems (efficiently) that cannot be solved (efficiently) without randomization
  - True in some computational models (e.g., for distributed algorithms)
  - Not clear in the standard sequential model

Contention Resolution

A simple example to start (from distributed computing / networking)

- Allows to introduce important concepts
- … and to repeat some basic probability theory

Setting:

- n processes, 1 resource
  (e.g., communication channel, shared database, …)
- There are time slots $1,2,3, \ldots$
- In each time slot, only one process can access the resource
- All processes need to regularly access the resource
- If process i tries to access the resource in slot t:
  - Successful iff no other process tries to access the resource in slot t

Algorithm

Algorithm Ideas:

- Accessing the resource deterministically seems hard
  - need to make sure that processes access the resource at different times
  - or at least: often only a single process tries to access the resource
- **Randomized solution:**
  In each time slot, each process tries with probability p.

Analysis:

- How large should p be?
- How long does it take until some process x succeeds?
- How long does it take until all processes succeed?
- What are the probabilistic guarantees?

Analysis

Events:

- $\mathcal{A}_{x,t}$: process x tries to access the resource in time slot t
  - Complementary event: $\overline{\mathcal{A}_{x,t}}$

$$\mathbb{P}(\mathcal{A}_{x,t}) = p, \qquad \mathbb{P}(\overline{\mathcal{A}_{x,t}}) = 1 - p$$

- $\mathcal{S}_{x,t}$: process x is successful in time slot t

$$\mathcal{S}_{x,t} = \mathcal{A}_{x,t} \cap \left(\bigcap_{y \neq x} \overline{\mathcal{A}_{y,t}} \right)$$

x is successful if
- x tries to access resource **and**
- no other process tries to access resource

- **Success probability** (for process x):

$$\mathbb{P}(\mathcal{S}_{x,t}) = \mathbb{P}(\mathcal{A}_{x,t}) \cdot \prod_{y \neq x} \mathbb{P}(\overline{\mathcal{A}_{y,t}}) = p \cdot (1-p)^{n-1}$$

choose p that maximizes $\mathbb{P}(\mathcal{S}_{x,t})$

Fixing p

$$\lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n = e^x$$

- $\mathbb{P}(\mathcal{S}_{x,t}) = p(1-p)^{n-1}$ is maximized for

$$\boldsymbol{p = \frac{1}{n}} \quad\Longrightarrow\quad \mathbb{P}(\mathcal{S}_{x,t}) = \frac{1}{n}\underbrace{\left(1-\frac{1}{n}\right)^{n-1}}_{\text{converges to } 1/e \text{ for } n\to\infty} \approx \frac{1}{en}$$

- **Asymptotics:**

$$\text{For } n \geq 2:\quad \frac{1}{4} \leq \left(1-\frac{1}{n}\right)^n < \frac{1}{e} < \left(1-\frac{1}{n}\right)^{n-1} \leq \frac{1}{2}$$

- **Success probability:**

$$\boldsymbol{\frac{1}{en} < \mathbb{P}(\mathcal{S}_{x,t}) \leq \frac{1}{2n}}$$

Time Until First Success

$q \approx \frac{1}{en}$

$q := \mathbb{P}(\mathcal{S}_{x,t}) = p(1-p)^{n-1}$

Random Variable T_x:

- $T_x = t$ if proc. x is successful in slot t for the first time

- **Distribution:**

$$\mathbb{P}(T_x = 1) = q,\ \mathbb{P}(T_x = 2) = (1-q) \cdot q, \dots$$

$$\mathbb{P}(T_x = t) = (1-q)^{t-1} \cdot q$$

- T_x is geometrically distributed with parameter

$$q = \mathbb{P}(\mathcal{S}_{x,t}) = \frac{1}{n}\left(1 - \frac{1}{n}\right)^{n-1} > \frac{1}{en}.$$

- **Expected time** until first success:

$$\mathbb{E}[T_x] := \sum_{t=1}^{\infty} t \cdot \mathbb{P}(T_x = t) = \frac{1}{q} < en$$

Time Until First Success

$$\left(1-\frac{1}{en}\right)^t < e^{-t/en}$$

Failure Event $\mathcal{F}_{x,t}$: Process x does not succeed in time slots $1, \dots, t$

$$\mathcal{F}_{x,t} = \bigcap_{r=1}^{t} \overline{\mathcal{S}_{x,r}}$$

$$\left(1-\frac{1}{en}\right) \le e^{-1/en}$$

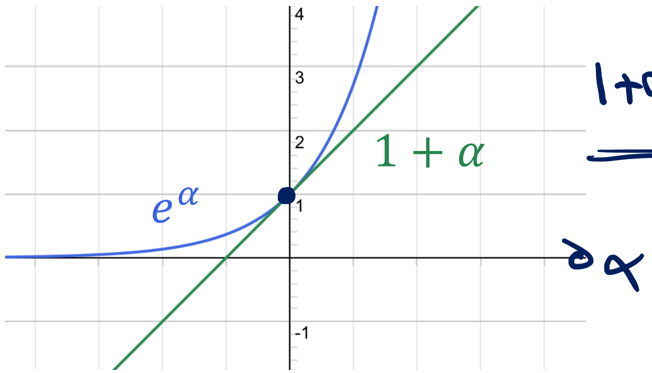
$$\alpha = -\frac{1}{en}$$

- The events $\mathcal{S}_{x,t}$ are independent for different t:

$$\mathbb{P}(\mathcal{F}_{x,t}) = \mathbb{P}\left(\bigcap_{r=1}^{t} \overline{\mathcal{S}_{x,r}}\right) = \prod_{r=1}^{t} \mathbb{P}\left(\overline{\mathcal{S}_{x,r}}\right) = \left(1 - \mathbb{P}(\mathcal{S}_{x,1})\right)^t = (1-q)^t$$

- We know that $\mathbb{P}(\mathcal{S}_{x,r}) > 1/en$:

$$\mathbb{P}(\mathcal{F}_{x,t}) < \underbrace{\left(1 - \frac{1}{en}\right)}_{1 - 1/en \,\le\, e^{-1/en}}{}^{t} \le e^{-t/en}$$



$$1+\alpha \le e^{\alpha}$$

Time Until First Success

$e^{-\frac{c\,en\ln n}{en}}$ $\qquad e^{c\ln n} = (e^{\ln n})^c$

No success by time t: $\mathbb{P}(\mathcal{F}_{x,t}) < e^{-t/en}$

- $t = \lceil en \rceil$: $\qquad \mathbb{P}(\mathcal{F}_{x,t}) < 1/e$
  - Generally, if $t = \Theta(n)$: constant success probability
- $t = \lceil c \cdot en \cdot \ln n \rceil$: $\quad \mathbb{P}(\mathcal{F}_{x,t}) < 1/e^{c \cdot \ln n} = 1/n^c$

$e^{c \cdot \ln n} = \left(e^{\ln n}\right)^c = n^c$

- For success probability $1 - 1/n^c$, we therefore need $t = \Theta(n \log n)$.
- We say that x succeeds **with high probability** in $O(n \log n)$ time.

With probability $\geq 1 - \frac{1}{n^c}$ for any constant $c > 0$.

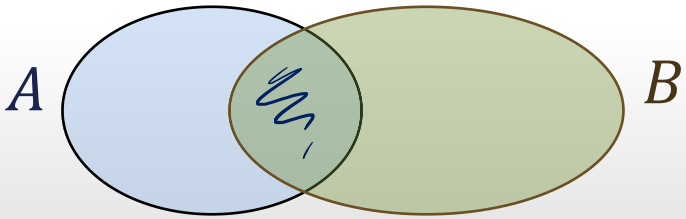
Choice of c only affects the hidden constant in the big-O notation.

Time Until All Processes Succeed

Event $\mathcal{F}_t$: some process has not succeeded by time t

$$\mathcal{F}_t = \bigcup_{x=1}^{n} \mathcal{F}_{x,t}$$

$$\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B)$$
$$\leq \mathbb{P}(A) + \mathbb{P}(B)$$



Union Bound: For events $\mathcal{E}_1, \dots, \mathcal{E}_k$,

$$\mathbb{P}\left(\bigcup_{x}^{k} \mathcal{E}_x\right) \leq \sum_{x}^{k} \mathbb{P}(\mathcal{E}_x)$$

$\mathbb{P}(\mathcal{F}_{x,t}) < e^{-t/en}$

Probability that not all processes have succeeded by time t:

$$\mathbb{P}(\mathcal{F}_t) = \mathbb{P}\left(\bigcup_{x=1}^{n} \mathcal{F}_{x,t}\right) \leq \sum_{x=1}^{n} \mathbb{P}(\mathcal{F}_{x,t}) < n \cdot e^{-t/en}.$$

Time Until All Processes Succeed

Claim: With high probability, all processes succeed in the first $O(n \log n)$ time slots.

Proof:

- $\mathbb{P}(\mathcal{F}_t) < n \cdot e^{-t/en}$
- Set $t = \lceil (c+1) \cdot en \cdot \ln n \rceil$

$$\mathbb{P}(\mathcal{F}_t) < n \cdot e^{-\frac{(c+1)\cdot en \cdot \ln n}{en}} = n \cdot e^{-(c+1)\cdot \ln n} = n \cdot \frac{1}{n^{c+1}} = \frac{1}{n^c}$$

Remarks:

- $\Theta(n \log n)$ time slots are necessary for all processes to succeed even with reasonable (constant) probability
- $\Theta(n \log n)$ time slots are also necessary in expectation for all processes to succeed at least once.

Primality Testing

$n = a \cdot b$ $\quad 1, \dots, n$ $\quad \underbrace{(1+o(1))\frac{n}{\ln n}}$

Problem: Given a natural number $n \geq 2$, is n a prime number?

Simple primality test:

1. **if** n is even **then**
2. **return** $(n = 2)$
3. **for** $i \coloneqq 1$ **to** $\lfloor \sqrt{n}/2 \rfloor$ **do**
4. **if** $2i + 1$ divides n **then**
5. **return false**
6. **return true**

- **Running time:** $O(\sqrt{n})$

If n is not prime, one of the prime factors p is $p \leq \lfloor \sqrt{n} \rfloor$:

$$2i + 1 \leq \lfloor \sqrt{n} \rfloor \Longrightarrow i \leq \left\lfloor \frac{\sqrt{n}}{2} \right\rfloor$$

Size of the input $O(\log n)$ bits:

$\sqrt{n}$ is *exponential* in the size of the input.

A Better Algorithm?

- How can we test primality efficiently?
  - We need a little bit of basic number theory…

$\mathbb{Z}_p^* = \{1, \dots, p-1\}$, multiplication mod p

Square Roots of Unity: In $\mathbb{Z}_p^*$, where p is a prime, the only solutions to the equation $x^2 \equiv 1 \pmod{p}$ are $x \equiv \pm 1 \pmod{p}$

$$x^2 \equiv 1 \pmod{p}$$
$$x^2 - 1 \equiv 0 \pmod{p}$$
$$(x+1)\cdot(x-1) \equiv 0 \pmod{p}$$

$\Rightarrow$ $\boldsymbol{(x+1)\cdot(x-1) = c \cdot p}$

for an integer c

$\Downarrow$

p is a prime factor of $x+1$ or of $x-1$:

$x + 1 \equiv 0 \pmod{p}$ or
$x - 1 \equiv 0 \pmod{p}$

$\Leftarrow$

Not true if p is not prime:
$p = 15, \quad x = 4$
$x^2 = 16 \equiv 1 \pmod{15}$

- If we find an $x \not\equiv \pm 1 \pmod{n}$ such that $x^2 \equiv 1 \pmod{n}$, we can conclude that n is not a prime.

Algorithm Idea

$a^p \equiv a \pmod p$

Claim: Let $p > 2$ be a prime such that $p - 1 = 2^s d$ for an integer $s \geq 1$ and some odd integer $d \geq 1$. Then for all $a \in \mathbb{Z}_p^*$,

$$a^d \equiv 1 \pmod p \ \text{ or } \ a^{2^r d} \equiv -1 \pmod p \ \text{ for some } 0 \leq r < s.$$

Recall that $x^2 \equiv 1 \pmod p \Leftrightarrow x \equiv -1, 1 \pmod p$

Proof:

- **Fermat's Little Theorem:** For every prime p and all $a \in \mathbb{Z}_p^*$: $a^{p-1} \equiv 1 \pmod p$
- Consider $x_0, x_1, \dots, x_s$, where $x_i = a^{\delta_i}$ for $\delta_i = \frac{p-1}{2^i} = 2^{s-i} \cdot d$

$a^{2^{s-i} \cdot d}$

$$\underbrace{\delta_0 = p - 1}_{= 2^s \cdot d} \quad \underbrace{\delta_1 = \frac{p-1}{2}}_{= 2^{s-1} \cdot d} \quad \underbrace{\delta_2 = \frac{p-1}{4}}_{= 2^{s-2} \cdot d} \quad \cdots \quad \underbrace{\delta_{s-1} = \frac{p-1}{2^{s-1}}}_{= 2 \cdot d} \quad \underbrace{\delta_s = \frac{p-1}{2^s}}_{= d}$$

$\cdot 2$ $\quad \cdot 2$ $\quad \cdot 2$

- $\forall i < s : x_i = x_{i+1}^2$, thus $x_i \equiv 1 \pmod p \Longrightarrow x_{i+1} \equiv -1, 1 \pmod p$
- Fermat's Little Theorem $\Longrightarrow x_0 \equiv 1 \pmod p$

$x_0 = 1 \to x_1 \in \{-1, 1\}$

$x_1 = 1 \to x_2 \in \{-1, 1\}$

- Thus: $\forall i \leq s : x_i \equiv 1 \pmod p$ or $\exists i \leq s : x_i \equiv -1 \pmod p$. *(which directly implies the claim.)*

Primality Test

We have: If n is an odd prime and $n - 1 = 2^s d$ for an integer $s \geq 1$ and an odd integer $d \geq 1$. Then for all $a \in \{1, \dots, n-1\}$,

$$a^d \equiv 1 \pmod{n} \ \textbf{ or } \ a^{2^r d} \equiv -1 \pmod{n} \ \text{ for some } 0 \leq r < s. \qquad (*)$$

Idea: If we find an $a \in \{1, \dots, n-1\}$ such that

$$a^d \not\equiv 1 \pmod{n} \ \textbf{ and } \ a^{2^r d} \not\equiv -1 \pmod{n} \ \text{ for all } 0 \leq r < s, \qquad \neg(*)$$

we can conclude that n is not a prime.

- For every odd composite $n > 2$, at least $^3/_4$ of all $a \in \{2, \dots, n-2\}$ satisfy condition $\neg(*)$.
- How can we find such a *witness* a efficiently?

Idea: pick a at random.

Miller-Rabin Primality Test

- Given a natural number $n \geq 2$, is n a prime number?

Miller-Rabin Test:

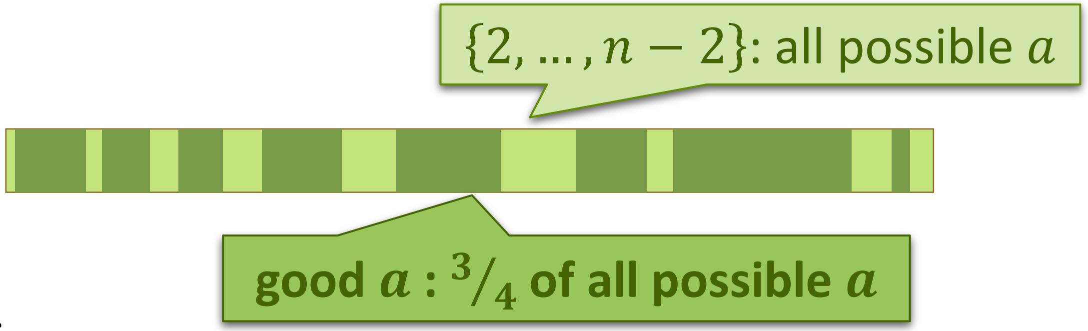
1. **if** n is even **then return** $(n = 2)$
2. compute s, d such that $n - 1 = 2^s d$; d odd
3. choose $a \in \{2, \dots, n-2\}$ uniformly at random;
4. $x \coloneqq a^d \bmod n$;
5. **if** $x = 1$ **or** $x = n - 1$ **then return probably prime;**
6. **for** $r \coloneqq 1$ **to** $s - 1$ **do**
7. $\quad x \coloneqq x^2 \bmod n$;
8. $\quad$ **if** $x = n - 1$ **then return probably prime;**
9. **return composite;**

$(*)$ holds

Analysis

Theorem:

- If n **is prime**, the Miller-Rabin test **always** returns **probably prime**.
- If n **is composite**, the Miller-Rabin test returns **composite** with **probability at least** $^3/_4$.



Proof:

- If n is prime, the test works for all values of a
- If n is composite, we need to pick a good witness a

Corollary: If the Miller-Rabin test is repeated k times, it fails to detect a composite number n with probability at most 4^{-k}.

Running Time

Cost of Modular Arithmetic:

- Representation of a number $x \in \mathbb{Z}_n$: $O(\log n)$ bits
- Cost of adding two numbers $x + y \bmod n$:

Time: $O(\log n)$

- Cost of multiplying two numbers $x \cdot y \bmod n$:
  - Done naively, this takes time $O(\log^2 n)$
  - It's like multiplying degree $O(\log n)$ polynomials
    → use FFT to compute $z = x \cdot y$

Time: $O(\log n \cdot \log\log n \cdot \log\log\log n)$

Running Time

a^d

$x^1 = x$

$x^{10} = (x^1)^2$

$x^{101} = x^{100} \cdot x = (x^{10})^2 \cdot x$

Cost of exponentiation x^d mod n:

- Can be done using $O(\log d)$ multiplications
- Base-2 representation of d: $d = \sum_{i=0}^{\lfloor \log_2 d \rfloor} d_i 2^i$
- **Fast exponentiation:**

  1. $y := 1;$
  2. **for** $i := \lfloor \log_2 d \rfloor$ **to** 0 **do**
  3. $\quad y := y^2 \bmod n;$
  4. $\quad$**if** $d_i = 1$ **then** $y := y \cdot x \bmod n;$
  5. **return** $y;$

- **Example:** $d = 22 = 10110_2$

1

$$x^{22} = \left(\left(\left((1^2 \cdot x)^2\right)^2 \cdot x\right)^2 \cdot x\right)^2$$

Running Time

Theorem: One iteration of the Miller-Rabin test can be implemented with running time $O(\log^2 n \cdot \log\log n \cdot \log\log\log n)$. $= \tilde{O}(\log^2 n)$

1. **if** n is even **then return** $(n = 2)$
2. compute s, d such that $n - 1 = 2^s d$;
3. choose $a \in \{2, \dots, n-2\}$ uniformly at random;
4. $x \coloneqq a^d \bmod n$;
5. **if** $x = 1$ **or** $x = n - 1$ **then return probably prime;**
6. **for** $r \coloneqq 1$ **to** $s - 1$ **do**
7. $\quad x \coloneqq x^2 \bmod n$;
8. $\quad$ **if** $x = n - 1$ **then return probably prime;**
9. **return composite;**

Time $O(\log n)$

$O(\log d) = O(\log n)$ multiplications

$s = O(\log n)$ iterations

1 multiplication per iteration

$O(\log n)$ multiplications $\Longrightarrow$ time $O(\log^2 n \cdot \log\log n \cdot \log\log\log n)$

Deterministic Primality Test

- If a conjecture known as the generalized Riemann hypothesis (GRH) is true, the Miller-Rabin test can be turned into a polynomial-time, deterministic algorithm

  → It is then sufficient to try all $a \in \{1, \dots, 2\ln^2 n\}$

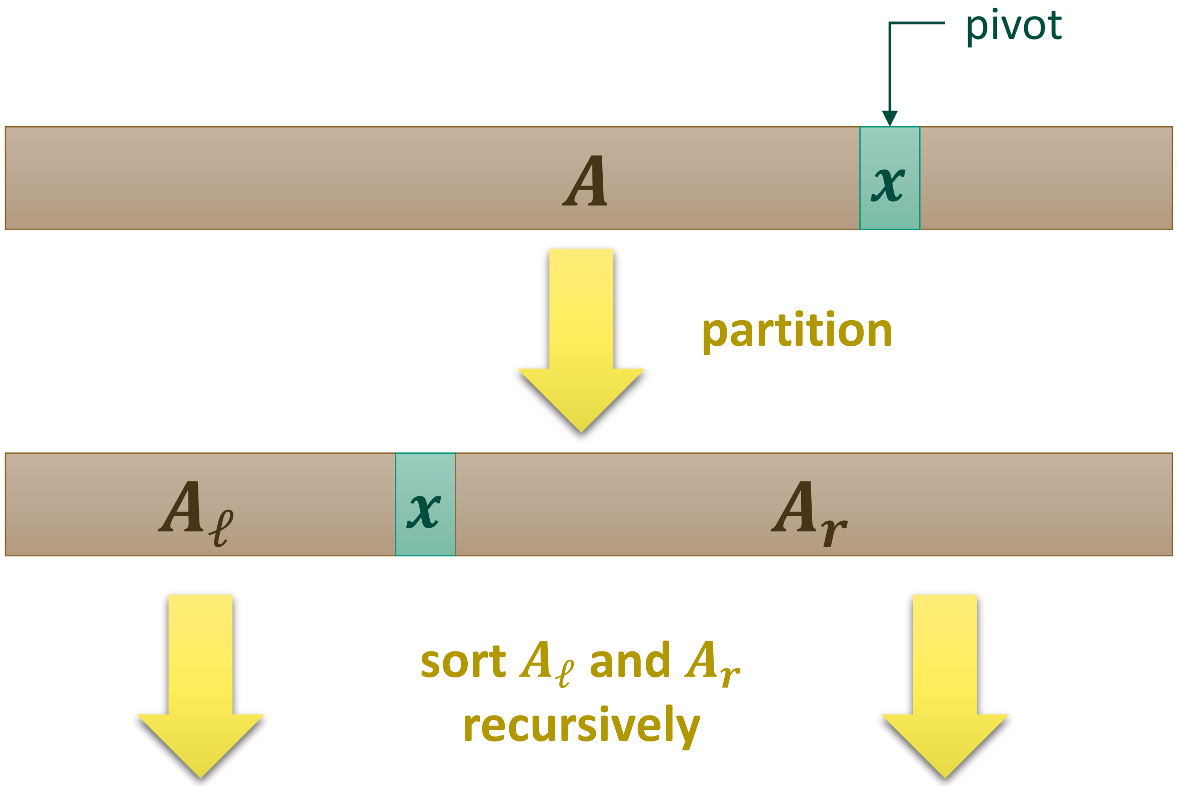
- It has long not been proven whether a deterministic, polynomial-time algorithm exists

hides factors polynomial in $\log\log n$

- In 2002, Agrawal, Kayal, and Saxena gave an $\tilde{O}(\log^{12} n)$-time deterministic algorithm
  - Has been improved to $\tilde{O}(\log^6 n)$

- In practice, the randomized Miller-Rabin test is still the fastest algorithm

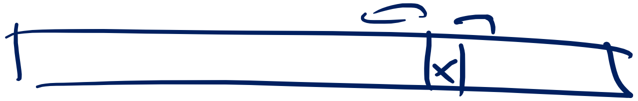
Randomized Quicksort



Randomized Quicksort:

pick pivot uniformly at random

Randomized Quicksort Analysis



Randomized Quicksort: pick uniform random element as pivot

Running Time of sorting n elements:

- Let us just count the number of comparisons
- In the partitioning step, all $n-1$ non-pivot elements have to be compared to the pivot

- **Number of comparisons:**

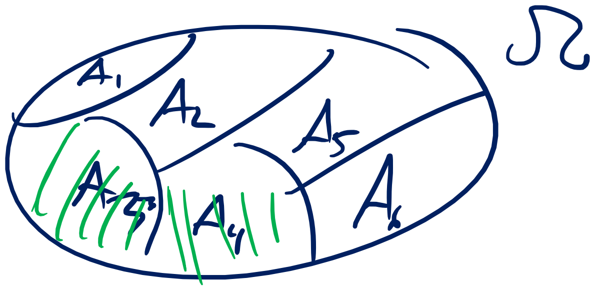
depends on choice of pivot

$$\boldsymbol{n-1} \;+\; \textbf{\#comparisons in recursive calls}$$

- **If rank of pivot is r:** recursive calls with $\boldsymbol{r-1}$ and $\boldsymbol{n-r}$ elements

$1, 2, 3, \dots,$ $\quad r-1, r, r+1, \dots,$ $\quad n-1, n$

Law of Total Expectation



- Given a random variable X and
- a set of events $A_1, \dots, A_k$ that partition Ω
  - E.g., for a second random variable Y, we could have $A_i \coloneqq \{\omega \in \Omega : Y(\omega) = i\}$

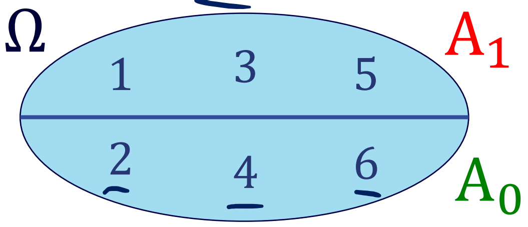
Law of Total Expectation

$$\mathbb{E}[X] = \sum_{i=1}^{k} \mathbb{P}(A_i) \cdot \mathbb{E}[X \mid A_i] = \sum_{y} \mathbb{P}(Y = y) \cdot \mathbb{E}[X \mid Y = y]$$

Example:

- X: outcome of rolling a die
- $A_0 = \{X \text{ is even}\}$, $A_1 = \{X \text{ is odd}\}$

Clearly: $\mathbb{E}[X] = \frac{1+2+3+4+5+6}{6} = 3.5$



$$\mathbb{E}[X] = \underbrace{\underbrace{\mathbb{E}[X|A_0]}_{=4} \cdot \underbrace{\mathbb{P}(A_0)}_{=1/2} + \underbrace{\mathbb{E}[X|A_1]}_{=3} \cdot \underbrace{\mathbb{P}(A_1)}_{=1/2}}_{=3.5}$$

Randomized Quicksort Analysis

Random variables:

- C: total number of comparisons (for a given array of length n)
- R: rank of first pivot
- C_ℓ, C_r: number of comparisons for the 2 recursive calls

Linearity of Expectation:

$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$

$$\mathbb{E}[C] = \mathbb{E}[n - 1 + C_\ell + C_r] = n - 1 + \mathbb{E}[C_\ell] + \mathbb{E}[C_r]$$

Law of Total Expectation:

$$\mathbb{E}[C] = \sum_{r=1}^{n} \mathbb{P}(R = r) \cdot \mathbb{E}[C | R = r]$$

$$= \sum_{r=1}^{n} \mathbb{P}(R = r) \cdot (n - 1 + \mathbb{E}[C_\ell | R = r] + \mathbb{E}[C_r | R = r])$$

Expected #comparisons to sort array of length n

Expected #comparisons to sort array of length $r - 1$

Expected #comparisons to sort array of length $n - r$

Randomized Quicksort Analysis

We have seen that:

$$\mathbb{E}[C] = \sum_{r=1}^{n} \underbrace{\mathbb{P}(R = r)}_{= 1/n} \cdot \left(n - 1 + \mathbb{E}[C_\ell | R = r] + \mathbb{E}[C_r | R = r]\right)$$

Define: $\boldsymbol{T(n)}$: expected number of comparisons when sorting n elements

$$\mathbb{E}[C] = T(n)$$
$$\mathbb{E}[C_\ell | R = r] = T(r - 1)$$
$$\mathbb{E}[C_r | R = r] = T(n - r)$$

Recursion:

$$\boldsymbol{T(n) = \sum_{r=1}^{n} \frac{1}{n} \cdot \left(n - 1 + T(r - 1) + T(n - r)\right)}$$

$$\boldsymbol{T(0) = T(1) = 0}$$